Note

A Study on Automatic Classification of Novice Homework Programs with Reasons

HARUO KAWASAKI*

(Received 14 Feburaru 2006)

Abstarct: We propose a new system NESCA that gives support to evaluation of novice homework programs. Evaluating students' programs on programming lectures is a hard work for teachers. Then, we intend our system to be used by teachers of computer programming lectures to evaluate novice programs. Understanding whole tendencies and characteristics of students' programs is important for teachers. NESCA classifies novice programs by use of classification information about assignment statements as a result of inferring programmers' intention before teachers' evaluation. Some systems that classify novice programs by statistical methods analyze the whole tendencies and characteristics. However, they cannot explain reasons for the classification. Our system presupposes little knowledge for the programs and classifies the programs with the reasons for the classification. Since students send many home work programs recently by networks, our system is an effective tool and preprocessor of program evaluation for teachers.

Keywords: program understanding, programming knowledge, modeling programs and programmers

1 Introduction

In this paper, we propose a new program evaluation support system, NESCA, for novices. The situation in which the system is used may be summarized as follows. Teachers of computer programming courses give students programming problems at their lectures and students make the programs as home works. In many cases the programs presented by students are quite similar. Sometimes, some of them are full copies of others.

In such situations, if we have a support system that evaluates and classifies programs according some reasons, it will be useful in the following ways. If the system is able to analyze results of classifications before teachers' evaluation, teachers will understand characters and tendencies of students' programs. Furthermore, teachers give students a warning of not to copy programs indirectly by announcing the results of analysis. Therefore, such a system will save much of the teachers' labor.

Many automatic plagiarism detection system in programming assignments has been developed. There are two major types of systems in this field, one is the attribute-counting type systems [1] and the other one is structure metric type system [2]. And a different type of system is presented by using matrices obtained from transformed structure trees of programs.

Many of these systems classify students' programs into clusters by statistical similarity

* Center for Information Science, Kokushikan University, 4–28–1 Setagaya, Setagaya-ku, Tokyo 154–8515, Japan
e-mail: kawasaki@kokushikan.ac.jp

[3]. The system can use already available statistical tools and programs for the analysis of programs, and also graphic tools for the presentation of the results of the analysis. However, these systems tend to generate many clusters, and consequently make it difficult to understand the reasons of the classification and the characters of the clusters.

Little attempts have been done for classifying programs with reasons. Although program understanding system [4], [5] are useful for novice programs evaluating support systems, a large amount of presupposed knowledge on model solution and buggy programs is needed. The effort to presuppose is too heavy a burden for teachers.

To overcome these shortages, we propose NESCA. NESCA classifies novice programs using information about computational purpose networks [6] before teachers' evaluation. A computational purpose network represents programmer's intention or plans for goals and sub-goals. Our system understands characters of classes and gives reasons for the classification. We construct a prototype system using Common LISP, Visual CLisp and ZERO [7].

2 Overview of Our System

We have introduced the inferring programmers' intention system COSMO based on a classification of assignment statements [6]. The system treats a subset of PASCAL-like programs that have assignment statements, input statements, output statements, conditional statements and while loops. We presume that input statements and output statements are special assignment statements.

COSMO constructs a dependence graph of a program using dependence analysis of the program. This is a directed graph whose vertices represent assignment statements in programs. The vertices are connected by the flow dependence edges.

Furthermore, member information as to which parts of the program the assignment statements are located in is added to the dependence graph of the program. COSMO infers the computational purpose assignment statements based on a classification of assignment statements. We consider that computational purpose assignment statements execute and conclude the intended purpose computation in loops or programs.

COSMO is a subsystem of NESCA. We introduce a similarity of programs by use of classification information about assignment statements and classify novice programs according to the similarity.

In this paper, we redefine the dependence graph by using definitions of labeled graph. In Fig. 1 we show the general flow of our system, NESCA. NESCA classify novice programs by the following 6 phases.

Phase 1: Flow Dependence Graph of a program (FDG) construction
Phase 2: Inferring programmers' intention by COSMO and construction of c-graph
Phase 3: Construction of c-p-networks
Phase 4: Grouping programs by using equivalence-check of c-p-networks
Phase 5: Giving reasons for classification by c-p-network information

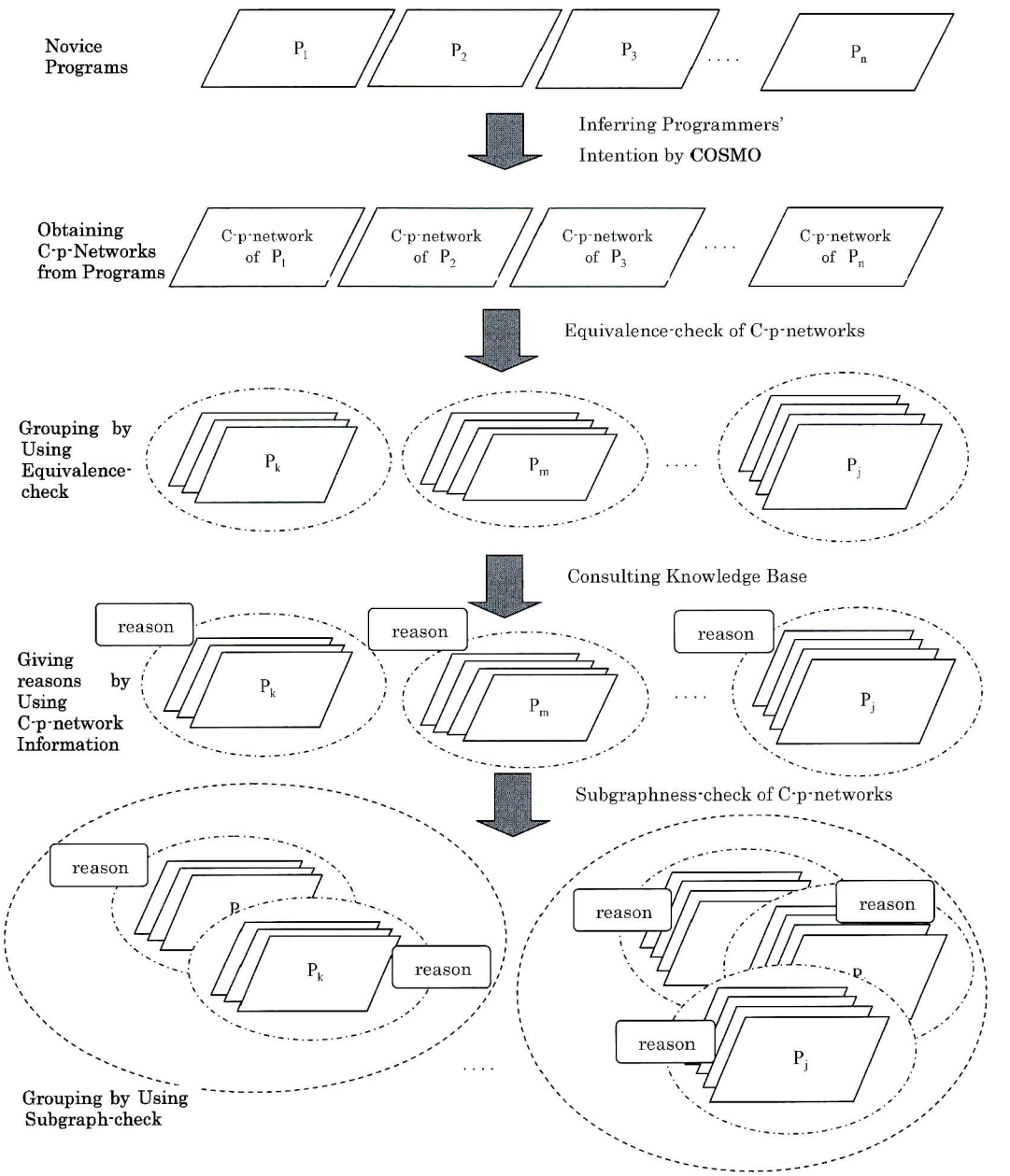


Fig. 1 General Flow of NESCA

Phase 6: Grouping program clusters by using subgraph-check of c-p-networks

At first, we explain some important terms in the following.

[Program] Our system treats a subset of PASCAL-like programs that contain assignment statements, input statements, output statements, conditional statements and while state-

ments. We represent programs by using PAD and attach names to boxes excluding conditional statements and while statements in PAD diagrams.

[Alphabetically variant program] An alphabetically variant program is a program obtained only by systematically renaming of some variables that exist in the original program.

[Dependence relation] Let s_1 and s_2 be statements in a program. When the following conditions are all satisfied, we say that a *flow dependence* from statement s_1 to statement s_2 by a variable v exists. This relation is denoted by $s_1 \rightarrow s_2$:

1. s_1 defines v, and
2. s_2 refers v, and
3. at least one execution path from s_1 to s_2 without redefining v exists.

However about the third condition, we exclude paths that do not pass thorough blocks of while-loops because of not satisfying loop predicates of while statements.

[Member information]

Programs are divided into some blocks by while statements in general. We want to know which parts of the blocks the assignment statements are located. We call this information of divided blocks as member information. The member information is obtained by the control dependency graphs.

At first, we define the Flow Dependence Graph (FDG) of a program. A FDG is a vertex labeled graph whose vertices represent names attached to boxes in PAD diagrams. Labels attached to vertices represent statements excluding conditional statements and while statements in a program, and whose edges denote flow dependence relations between statements attached to vertices.

[Definition 1] The FDG of program P is a labeled graph g_P and a triple $g_P = (V, E, \beta_0)$ where

(1). let $\Pi_1, \Pi_2, \ldots, \Pi_n$ be divided blocks of P and m_i be the total number of assignment statements, input statements and output statements in Π_i $(1 \leq i \leq n)$.
Let where. is a corresponding vertices set to .
Let $V = \cup_{i=1}^{n} V_i$ where $V_i = \{v_{ij} \mid j = 1, 2, \ldots, m_i\}$. V_i is a corresponding vertices set to Π_i $(1 \leq i \leq n)$.
V is a set of vertices and each vertex v_{ij} represents a name attached to a box in a PAD diagram of P.

(2). Let L_S be a set of vertices labels. Vertices labels represent statements excluding conditional statements and while statements in P.
Let β_0: $V \rightarrow L_S$ be a function labeling the vertices.
$\beta_0(v) = s$ $(v \in V, s \in L_S)$ iff v is a name of a box in PAD diagram of P and s is a statement attached to the box.

(3). Let E is a set of edges over $V \times V$.

$(v, v') \in E$ $(\forall v, v' \in V)$ iff $s \rightarrow s'$ and $\beta_0(v) = s$, $\beta_0(v') = s'$.

For example, we attach names to each assignment statements of a mean value program in Fig. 2. This program is divided into 3 blocks, Π_1, Π_2, Π_3.

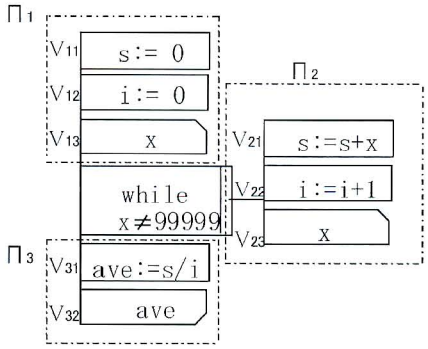


Fig. 2 A Mean Value Program

A subsystem COSMO classifies assignment statements into the control assignment statements (CNV), initial value assignment statements (INT), input assignment statements (IPT), output statements (OUT) and the computational purpose assignment statements (CMP).

[Definition 2] L_D is the classification information set of assignment statement and $L_D = \{INI, CMP, CNV, OUT, IPT, \varepsilon\}$. ε means that an assignment statement is not classified.

Then, we define a classification information added dependence graph obtained from program P (c-graph of P). C-graph obtained from P is a vertex labeled graph whose vertices represent names attached to boxes in PAD diagrams. Pairs of labels attached to vertices represent statements excluding conditional statements and while statements in P, and corresponding classification information sets. The edges denote *flow dependence* relations between statements attached to vertices.

[Definition 3] Let g_P be a FDG of program P and $g_P = (V, E, \beta_0)$. A classification information added dependence graph obtained from program P (abbr. c-graph of P) is a labeled graph d_p and a triple $d_p = (V, E, \beta)$ where

(1). V is the same vertices set of g_P.

(2). Let β: $V \rightarrow L_S \times 2^{L_D}$ be a function labeling the vertices.

$\beta(v) = (s, C)$ $(v \in V, s \in L_S, C \in 2^{L_D}, \beta_0(v) = s)$

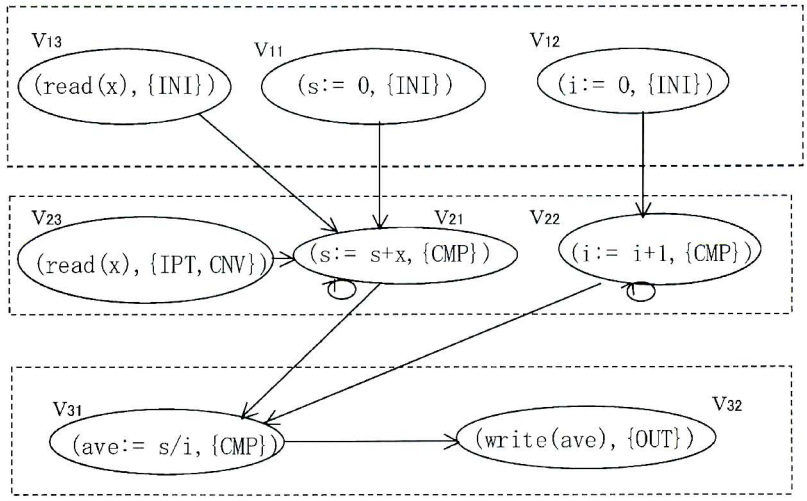
iff a name v is attached to a box in PAD diagram of P and s is a statement attached to the box. C is a classification information set of s obtained by COSMO. L_S is the same label set of g_P.

(3). Let E be a set of edges over $V \times V$.

$(v, v') \in E$ $(\forall v, v' \in V)$

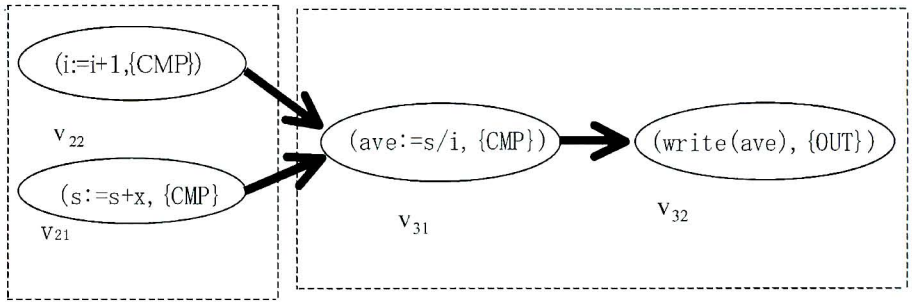
iff $s \rightarrow s'$ and the first element of $\beta(v)$ is s, the first element of $\beta(v')$ is s'.

[Definition 4] Let P be a program and d_p be a c-graph obtained from P. Let $\tilde{p} = (v_1, v_2, \ldots, v_n)$ be a vertices sequence of d_p. If d_p contains edges from vertex v_i to v_{i+1} for all i $(1 \leq i \leq n-1, n \geq 2)$, then we call $\tilde{p}$ a c-execution path from v_1 to v_n.



$d_p = (V, E, \beta)$
$V = \cup_{i=1}^{3} V_i$, $V_1 = \{v_{11}, v_{12}, v_{13}\}$, $V_2 = \{v_{21}, v_{22}, v_{23}\}$, $V_3 = \{v_{31}, v_{32}\}$
$E = \{(v_{13}, v_{21}), (v_{11}, v_{21}), (v_{12}, v_{22}), (v_{23}, v_{21}), (v_{21}, v_{21}), (v_{22}, v_{22}), (v_{21}, v_{31}), (v_{22}, v_{31}), (v_{31}, v_{32})\}$
$\beta(v_{11}) = (s:=0, \{INI\})$, $\beta(v_{12}) = (i:=0, \{INI\})$, $\beta(v_{13}) = (read(x), \{INI\})$
$\beta(v_{21}) = (s:=s+x, \{CMP\})$, $\beta(v_{22}) = (i:=i+1, \{CMP\})$, $\beta(v_{23}) = (read(x), \{IPT, CNV\})$
$\beta(v_{31}) = (ave:=s/i, \{CMP\})$, $\beta(v_{32}) = (write(ave), \{OUT\})$

Fig. 3 The Classification Informaton Added Dependence Graph of The Mean Value Program



$C_p = (\tilde{V}, \tilde{E}, \tilde{\beta})$
$\tilde{V}_1 = \phi$, $\tilde{V}_2 = \{v_{21}\ v_{22}\}$, $\tilde{V}_3 = \{v_{31}\ v_{32}\}$, $\tilde{V} = \tilde{V}_1 \cup \tilde{V}_2 \cup \tilde{V}_3$
$\tilde{E} = \{(v_{21}, v_{31}), (v_{22}, v_{31}), (v_{31}, v_{32})\}$
$\tilde{\beta}(v_{21}) = (s:=s+x, \{CMP\})$, $\tilde{\beta}(v_{22}) = (i:=i+1, \{CMP\})$
$\tilde{\beta}(v_{31}) = (ave:=s/i, \{CMP\})$, $\tilde{\beta}(v_{32}) = (write(ave), \{OUT\})$

Fig. 4 The Classification information Added Computational Purpose Network of the Mean Value Program

[Definition 5] Let P be a program. Let d_p be a c-graph of P and $d_p = (V, E, \beta)$. Let s and s' be statements of P and $v_1, v_2, \ldots, v_n \in V$ $(n \geq 2)$. Let $\tilde{p} = (v_1, v_2, \ldots, v_n)$ be a c-execution path from v_1 to v_n and $v_1 \neq v_n$. Let $\beta(v_1) = (s, C)$ and $\beta(v_n) = (s', C')$ $(C, C' \in 2^{L_D})$.

A pair of vertex (v_1, v_n) is called a selected edge of d_p. Also, v_1 and v_n are called selected

vertices of d_p iff one of the following two conditions are satisfied.

(1). CMP$\in$C, CMP$\in C'$ and there is no other vertex v_j ($1<j<n$, $n\geq 2$) in $\tilde{p}$, where CMP is a member of the second element of $\beta(v_j)$.

(2). CMP$\in$C, $C'=\{OUT\}$ and there is no other vertex v_j ($1<j<n$, $n\geq 2$) in $\tilde{p}$, where CMP is a member of the second element of $\beta(v_j)$.

Then, we define a classification information added computational purpose network of program P (c-p-network of P).

[Definition 6] Let d_p be a c-graph obtained from program P and $d_p=(V, E, \beta)$. A classification information added computational purpose network obtained from P (abbr. c-p-network of P) is a labeled graph c_p and a triple $c_p=(\tilde{V}, \tilde{E}, \tilde{\beta})$.

(1) $\tilde{V}$ is a subset of V and, $\tilde{V}$ is the set of all selected vertices of d_p

where $\tilde{V}=\cup_{i=1}^{n}\tilde{V}_i$, $V_i=\{v_{ij}\mid j=1, 2, \ldots, m_i'\wedge 0\leq m_i'\leq m_i\}$, $\tilde{V}_i\subset V_i$ ($1\leq i\leq n$).

(2). $\tilde{\beta}$: $\tilde{V}\rightarrow\tilde{L}_S\times\tilde{2}^{L_D}$ is a function labeling vertices and $\tilde{\beta}=\beta\mid\tilde{V}$, and $\tilde{L}_S\subset L_S$, $\tilde{2}^{L_D}\subset 2^{L_D}$.

(3). $\tilde{E}$ is a set of edges over $\tilde{V}\times\tilde{V}$ and $\tilde{E}=\{(v, v')\mid v$ and v' are selected vertices of $d_p\}$.

[Definition 7] Let P_1, P_2 be programs and let c_{p_1}, c_{p_2} be c-p-networks of P_1 and P_2. Let $c_{p_1}=(\tilde{V}_1, \tilde{E}_1, \tilde{\beta}_1)$ and $c_{p_2}=(\tilde{V}_2, \tilde{E}_2, \tilde{\beta}_2)$, $\tilde{V}_1=\cup_{i=1}^{n_1}\tilde{A}_i$, $\tilde{V}_2=\cup_{i=1}^{n_2}\tilde{B}_i$. Then, c_{p_1} is isomorphic to c_{p_2} by $\tilde{f}$ (abbr. $c_{p_1}\cong^{\tilde{f}} c_{p_2}$), iff there exists a bijective function $\tilde{f}$: $\tilde{V}_1\rightarrow\tilde{V}_2$, where

(1). $(a_{ij}, a_{pq})\in\tilde{E}_1\wedge i=p\Leftrightarrow(b_{ef}, b_{rs})\in\tilde{E}_2\wedge b_{ef}=\tilde{f}(a_{ij})$, $b_{rs}=\tilde{f}(a_{pq})$
$\wedge b_{ef}\in\tilde{B}_e$, $b_{rs}\in\tilde{B}_r\wedge e=r$

(2). $(a_{ij}, a_{pq})\in\tilde{E}_1\wedge i\neq p\Leftrightarrow(b_{ef}, b_{rs})\in\tilde{E}_2\wedge b_{ef}=\tilde{f}(a_{ij})$, $b_{rs}=\tilde{f}(a_{pq})$
$\wedge b_{ef}\in\tilde{B}_e$, $b_{rs}\in\tilde{B}_r\wedge e\neq r$

for any $a_{ij}\in\tilde{A}_i$, $a_{pq}\in\tilde{A}_p$.

[Definition 8] Let P_1, P_2 and P_2' be programs. Let P_2' be an alphabetically variant program of P_2. Let c_{p_1} be a c-p-network of P_1. Let c_{p_2} be a c-p-network of P_2 and let $c_{p_2'}$ be a c-p-network of P_2'. Let $c_{p_1}=(\tilde{V}_1, \tilde{E}_1, \tilde{\beta}_1)$, $c_{p_2}=(\tilde{V}_2, \tilde{E}_2, \tilde{\beta}_2)$, $c_{p_2'}=(\tilde{V}_2', \tilde{E}_2', \tilde{\beta}_2')$.
Then, c_{p_1} is equivalent to c_{p_2} (abbr. $c_{p_1}\equiv c_{p_2}$), iff $\tilde{\beta}_1=\tilde{\beta}_2'\circ\tilde{f}$ and $c_{p_1}\cong^{\tilde{f}} c_{p_2'}$.

[Definition 9] Let P_1, P_2 and P_2' be programs. Let P_2' be an alphabetically variant program of P_2. Let c_{p_1} be a c-p-network of P_1. Let c_{p_2} be a c-p-network of P_2 and let $c_{p_2'}$ be a c-p-network of P_2'. Let $c_{p_1}=(\tilde{V}_1, \tilde{E}_1, \tilde{\beta}_1)$, $c_{p_2}=(\tilde{V}_2, \tilde{E}_2, \tilde{\beta}_2)$, $c_{p_2'}=(\tilde{V}_2', \tilde{E}_2', \tilde{\beta}_2')$.
Then, c_{p_1} is called a subgraph of c_{p_2} (abbr. $c_{p_1}\subset c_{p_2}$) iff

(1). $\tilde{V}_1\subset\tilde{V}_2'$

(2). $\tilde{E}_1\subset\tilde{E}_2'\cap(\tilde{V}_1\times\tilde{V}_1)$

(3). $\tilde{\beta}_1=\tilde{\beta}_2'\mid\tilde{V}_1$

Thus, we define a similarity of programs by using equivalence of c-p-networks.

[Definition 10] Let P_1, P_2, be programs and let c_{p_1}, c_{p_2} be c-p-networks of P_1 and P_2.
P_1 is similar to P_2 iff $c_{p_1}\equiv c_{p_2}$.

3 Experiment

We construct a prototype system on the SUN workstation and a PC. In this section, we show an example by NESCA. We use Common LISP, Visual CLisp and ZERO for developing our system. Fig. 5 shows program information by using frame-type knowledge expression language ZERO. PROGRAM-slot expresses an input program. D-GRAPH-slot means a dependence graph of the input program and C-P-NETWORK-slot means a c-p-network of the program. And VERTEX-EDGE-slot shows number of vertices and edges of c-p-networks. Furthermore, PURPOSE-slot expresses purposes of programs. We treated a program that computed a mean value of input data. The experiment was performed for 16 students of an introductory programming course.

Fig. 6 shows a part of the result of this experiment. NESCA classified 16 programs into three groups (2 1), (3 2) and (4 3) by the use of information about number of vertices and edges of c-p-networks.

```
FRAME NAME:MVEX01F                FRAME TYPE:CLASS
A-KIND-OF          FRAME          ROOT
DDESCENDANTS       FLIST          NIL
DESCRIPTION        STRING         *UNDEFINED*
CREATED-BY         STRLIST        ("k100040" "18-1-2005 17:49:10")
MODIFIED-BY        STRLIST        ("k100040" "30-1-2005 20:20:02")
PURPOSE            STRLIST        ("model program" "mean value of input data")
VARIABLE           LIST           (S I N X H)
PROGRAM            LIST
 ((((A1 1) (S := 0)) ((A1 2) (I := 1))
                     ((A1 3) (READ { N })))
  ((WHILE I < = N) ((A2 1 W) (READ { X }))
                   ((A2 2 W) (S := S + X))
                   ((A2 3 W) (I := I + 1)))
  (((A3 1) (H := S / N)) ((A3 2) (WRITE { H }))
                         ((A3 3) (WRITE { N }))))
D-GRAPH            LIST
 ((((A1 1) (S := 0)) ((A2 2 W) (S := S + X)))
  (((A1 2) (I := 1)) ((A2 3 W) (I := I + 1)))
  (((A1 3) (READ { N })) ((A3 1) (H := S / N)))
  (((A1 3) (READ { N })) ((A3 3) (WRITE { N })))
  (((A2 1 W) (READ { X })) ((A2 2 W) (S := S + X)))
  (((A2 2 W) (S := S + X)) ((A3 1) (H := S / N)))
  (((A3 1) (H := S / N)) ((A3 2) (WRITE { H }))))
C-P-NETWORK        LIST
 ((((A2 2 W) (S := S + X) (CMP)) ((A3 1) (H := S / N) (CMP)))
   (((A3 1) (H := S / N) (CMP)) ((A3 2) (WRITE { H }) (OUT))))
VERTEX-EDGE        LIST           (3 2)
```

Fig. 5 Program Information by Using Frame-type Knowledge Expression

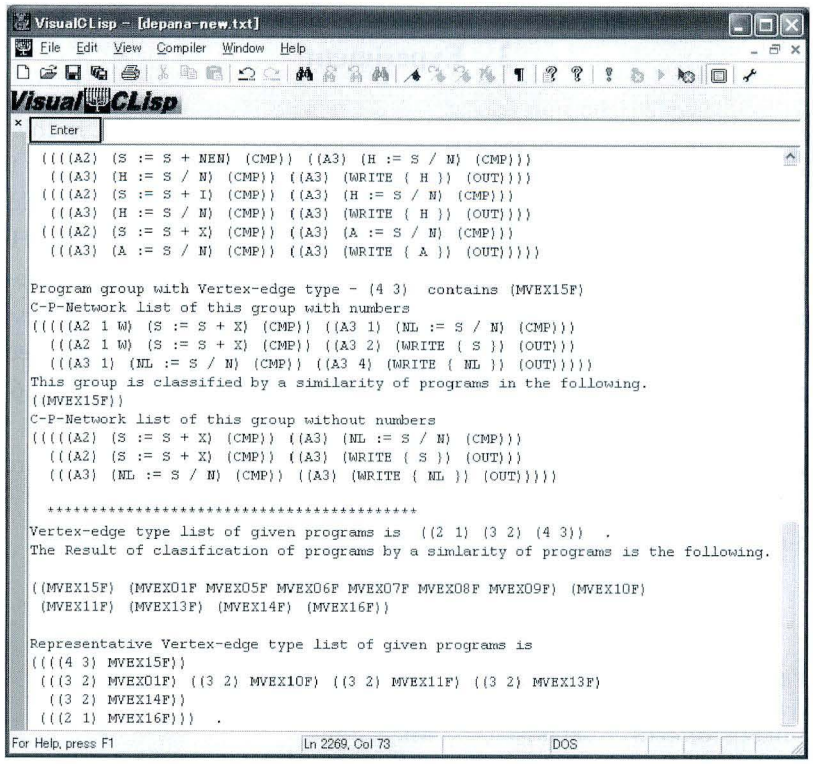


Fig. 6 Output Example of NESCA

4 Discussion and Conclusion

We propose a new program evaluating support system based on a classification of assignment statements. When a system does not have much presupposed knowledge, NESCA is an intelligent effective tool. We carry on more investigation into relation checking methods of classes.

References

[1] K. Ottenstein: "An Algorithmic Approach to the Detection and Prevention of Plagiarism", ACM SIGCSE Bull., 8, pp35–41, 1976

[2] M. Wise: "YAP3: Improved Detection of Similarities in Computer Program and Other Text", In Proc. 27th SIGCSE Technical Symp. On Computer Science Education, pp130–134, ACM Press, 1996

[3] M. Terada: "Detection and Clustering of Similar Programs, Proc. The 38-th Programming Symposium", pp 21–32, 1997, (in Japanese)

[4] R. Sekimoto, K. Kaijiri and M. Yamagata: "Development of the Report Preprocessing System Using Network for Educational Use", JSISE, Vol. 14, No. 5, pp217–222, 1998 (in Japanese)

[5] H. Ueno: "Knowledge Based Intelligent Programming Environment", J. IPS Japan, pp1280–1296, 1987

[6] H. Kawasaki: "Inferring Programmers' Intention by the Use of Context Structure Model of Programs", IEICE Trans. on Inf. And Sys., Vol. E83–D, No. 4 April, pp835–844, 2000

[7] H. Ito and H. Ueno: "ZERO: Frame+Prolog", Logic Programming '85, LNCS221, pp10–17, 1986